CSCI 136
Data Structures & Advanced Programming

Lecture 6
Fall 2017
Instructors: Bill & Bill
Last Time

- The class Object
  - Provides default toString() and equals() methods
- Example: Card Deck (Array/Vector versions)
- Associations and Vectors
Today’s Outline

- Associations
- Code Samples
  - WordFreq, Dictionary (Associations, Vectors)
- Generic Data Types
- Lab 2 Design and Strategies
- Vector Implementation
- Miscellany: Access Modifiers, Wrappers
- Condition Checking
  - Pre- and post-conditions, Assertions
Example: Word Counts

• Goal: Determine word frequencies in files
• Idea: Keep a Vector of (word, freq) pairs
  • When a word is read…
    • If it’s not in the Vector, add it with freq = 1
    • If it is in the Vector, increment its frequency
• How do we store a (word, freq) pair?
  • An Association
Associations

- Word → Definition
- Account number → Balance
- Student name → Grades

Google:
- URL → page.html
- page.html → \{a.html, b.html, …\} (links in page)
- Word → \{a.html, d.html, …\} (pages with Word)

In general:
- **Key** → **Value**
Association Class

- We want to capture the “key $\rightarrow$ value” relationship in a general class that we can use everywhere
- What type do we use for key and value instance variables?
  - Object!
  - We can treat any thing as an Object since all classes inherently extend Object class in Java…
class Association {
    protected Object key;
    protected Object value;

    //pre: key != null
    public Association (Object K, Object V) {
        Assert.pre (K!=null, "Null key");
        key = K;
        value = V;
    }

    public Object getKey() {return key;}
    public Object getValue() {return value;}
    public Object setValue(Object V) {
        Object old = value;
        value = V;
        return old;
    }

    // Continued on next slide....
public boolean equals(Object other) {
    if (other instanceof Association) {
        Association otherAssoc = (Association) other;
        return getKey().equals(otherAssoc.getKey());
    }
    else return false;
}

• Note: The actual structure package code does NOT do the instanceof check (but it should).
• Instead the method has a “pre-condition” comment that says the other must be a non-null Association!
Let’s Write WordFreq.java

- Goal: Determine word frequencies in files
- Idea: Keep a Vector of Associations between words and the number of occurrences
  - When a word is read…
  - If it’s new, add new Association(word, 1)
  - If it exists, update existing Association’s count
- How do we update a (word, freq) pair?
  - Vector get + Association set
  - Draw a picture!
WordFreq.java

- Uses a Vector
  - Each entry is an Association
  - Each Association is a (String, Integer) pair
- Notes:
  - Include structure5.*;
  - Can create a Vector with an initial capacity
  - Must cast the Objects removed from Association and Vector to correct type before using
Notes About Vectors

• **Primitive Types and Vectors**
  
  ```java
  Vector v = new Vector();
v.add(5);
  ```

  • This (technically) shouldn’t work! Can’t use primitive data types with vectors…they aren’t Objects!

  • Java is now smart about some data types, and converts them automatically for us -- called autoboxing

• **We used to have to “box” and “unbox” primitive data types:**

  ```java
  Integer num = new Integer(5);
v.add(num);
  ...
  Integer result = (Integer)v.get(0);
  int res = result.intValue();
  ```

• **Similar wrapper classes (Double, Boolean, Character) exist for all primitives**
protected Vector defs;
public Dictionary() {
    defs = new Vector();
}

public void addWord(String word, String def) {
    defs.add(new Association(word, def));
}

// post: returns the definition of word, or "" if not found.
public String lookup(String word) {
    for (int i = 0; i < defs.size(); i++) {
        Association a = (Association)defs.get(i);
        if (a.getKey().equals(word)) {
            return (String)a.getValue();
        }
    }
    return "";
}
```java
public static void main(String args[]) {
    Dictionary dict = new Dictionary();
    dict.addWord("perception", "Awareness of an object of thought");
    dict.addWord("person", "An individual capable of moral agency");
    dict.addWord("pessimism", "Belief that things generally happen for the worst");
    dict.addWord("philosophy", "Literally, love of wisdom.");
    dict.addWord("premise", "A statement whose truth is used to infer that of others");
}
```
Casting is DANGEROUS

- What limitations are associated with casting Objects as they are added and removed from Associations?
  - Errors cannot be detected by compiler
  - Must rely on runtime errors
  - Compiler complaints
Using Generic (Parameterized) Types

• Instead of casting Objects, Java supports using generic or parameterized data types (Read Ch 4)

• Instead of:

```java
Association a = new Association("Bill",(Integer) 97);
Integer grade = (Integer) a.getValue();  //Cast to String
```

• Use:

```java
Association<String, Integer> a =
    new Association<String, Integer>("Bill", (Integer) 97);
Integer grade = a.getValue();  //no cast!
```
Generic Association<K,V> Class

class Association<K,V> {
    protected K theKey;
    protected V theValue;

    //pre: key != null
    public Association (K key, V value) {
        Assert.pre (key != null, "Null key");
        theKey = key;
        theValue = value;
    }

    public K getKey() {return theKey;}
    public V getValue() {return theValue;}
    public V setValue(V value) {
        V old = theValue;
        theValue = value;
        return old;
    }
}

What About Generic Vectors?

• Instead of:
  Vector v = new Vector();  //Vector of Objects
  String word = (String)v.get(index);  //Cast to String

• Use:
  Vector<String> v = new Vector<String>();  //Vector of Strings
  String word = v.get(index);  //no cast!

• Or:
  Vector<Association<String, Integer>> v =
    new Vector<Association<String, Integer>>();
  int count = v.get(index).getValue();  //no cast!

• See GenWordFreq.java...

(Look at WordFreq.java with gen)
Lab 2

• Three classes:
  • Table.java
  • FrequencyList.java
  • WordGen.java

• Two Vectors of Associations

• toString() in Table and FrequencyList for debugging

• What are the key stages of execution?
  • Test code thoroughly before moving on to next stage

• Use WordFreq as example
Lab 2: Core Tasks

• FrequencyList
  • Vector< Association< Character, Integer > >
  • Add a letter
    • Is it a new letter or not?
    • UseindexOf for Vector class

• Pick a random letter based on frequencies
  • Let total = sum of frequencies in FL
  • generate random int r in range [0…total]
  • Find smallest k s.t r >= sum of first k frequencies
Lab 2: Core Tasks

• Table
  • Add a letter to a k-gram
    • Is it a new k-gram or not?
  • Pick a random letter given a k-gram
    • Find the k-gram then ask its FrequencyList to pick

• WordGen

• Convert input into (very long) String
  • Use a StringBuffer---see handout
Implementing Vectors

• A Vector holds an array of Objects
• Key difference is that the number of elements can grow and shrink dynamically
• How are they implemented in Java?
  • What instance variables do we need?
  • What methods? (start simple)
• We’ll focus on the generic version
• Let’s explore the implementation…. 
public class Vector<E> {
    private Object[] elementData; // Underlying array
    protected int elementCount; // Number of elts in Vector
    protected final static int defaultCapacity;
    protected int capacityIncrement; // How much to grow by
    protected E initialValue; // A default elt value
}

• Why Object[]?
  • Java restriction: Can’t use type variable, only actual type

• Why elementCount?
  • size won’t usually equal capacity

• Why capacityIncrement?
  • We’ll “grow” the array as needed
Basic Vector<E> Methods

```java
public class Vector<E> {
    public Vector() // Make a small Vector
    public Vector(int initCap) // Make Vector of given capacity
    public void add(E elt) // Add elt to (high) end of Vector
    public void add(int i, E elt) // Add elt at position i
    public E remove(E elt) // Remove (and return) elt
    public E remove(int i) // Remove (and return) elt at pos i
    public int capacity() // Return capacity
    public int size() // Return current size
    public boolean isEmpty() // Is size == 0?
    public boolean contains(E elt) // Is elt in Vector?
    public E get(int i) // Return elt at position i
    public E set(int i, E elt) // Change value at position i
    public int indexOf(E elt) // Return earliest position of elt
}
```
Class Vector : Basic Methods

- Much work done by few methods:
  - `indexOf(E elt, int i)` // find first occurrence of `elt` at/after pos. `i`
    - Used by `indexOf(E elt)`
    - remove methods use `indexOf(E elt)`
  - `firstElement()`, `lastElement()` use `get(int i)`
- Method names/functions in spirit of Java classes
  - `indexOf` has same behavior as for Strings
- Methods are straightforward except when array is full
- How do we add to a full Vector?
  - We make a new, larger array and copy values to it
Extending the Array

• How should we extend the array?
• Possible extension methods:
  • Grow by fixed amount when capacity is reached
  • Double array when capacity is reached
• How could we compare the two techniques?
  • Run speed tests?
    • Hardware/system dependent
  • Count operations!
• We’ll do this soon